Um Modelo de Fluxo de Dados e Respectiva Arquitetura

Claudio Roland Sonnenburg
Instituto de Estudos Avançados, CTA
Rod. dos Tamoios km 5,5, São José dos Campos, SP
email: sonnen@ieav.cta.br

Resumo

Este artigo tem por objetivo a descrição suscinta de mais um modelo de computação paralela e uma correspondente arquitetura onde programas descritos no modelo poderão executar com alto grau de paralelismo. Além do paralelismo que é exposto naturalmente, o modelo se presta para uma exploração adicional (significativa) de paralelismo, através da linearização parcial de estruturas iterativas, aliada à simples reestruturação dos grafos correspondentes. O modelo e a respectiva arquitetura se enquadram na classe genérica de "Arquiteturas Estáticas de Fluxo de Dados".

Abstract

In this paper we propose one more model of parallel computation and a corresponding architecture where programs described in the model can execute with high degree of concurrency. In addition to the parallelism that is exposed naturally when a program is translated to the model, the framework is very suitable for additional exploitation of parallelism through the combination of partial linearization of iterative structures and simple restructuring of the resulting graphs. The model and the architecture belong to the generic class of "Static Data Flow Architectures".

1. Introdução

Apesar do relativo esforço de pesquisa realizado nos últimos anos na área de Arquiteturas de Fluxo de Dados [1], [2], [3], [4], e das promessas de melhores performances que sempre caracterizaram a área, lamentavelmente ainda não apareceu no mercado uma máquina comercialmente viável baseada nestes conceitos. As razões atuais, na opinião deste autor, têm a ver com os seguintes fatos:

a) As relações preço/performance dos microprocessadores RISC têm sido sempre muito favoráveis, desde os primeiros lançamentos.

b) A expectativa de que as performances desses microprocessadores sejam quadruplicadas a cada três anos, com manutenção da compatibilidade, têm se materializado nos últimos anos.

Estes fatos estabelecem um cenário no qual fica muito difícil seguir em uma direção, no desenvolvimento de novas arquiteturas, que não inclua microprocessadores RISC como componentes básicos. Nenhum fabricante pode correr o risco de constatar que o nó da sua máquina paralela, quando finalmente pronta, esta gerações atrás dos microprocessadores RISC da época. Esta é, sem dúvida, a razão da proliferação dos multicomputadores e multiprocessadores baseados em microprocessadores RISC, dos quais o INTEL PARAGON, o TMC CM-5E, o CRAY T3D, o SGI POWER CHALLENGE, o KENDALL SQUARE KSR2 e o IBM SP2 são exemplos recentes.

Os efeitos desta situação também se manifestam na própria área de Fluxo de Dados, com o advento de uma linha de pesquisa que procura conciliar os conceitos da organização von Neumann (e a possibilidade de usar microprocessadores RISC) com Fluxo de Dados [5], [6].

Por outro lado, a experiência adquirida com máquinas vetoriais e o enorme acêrvo de software existente para esses computadores, que implica no requisito de compatibilidade com o passado, tornam os supercomputadores vetoriais uma outra forte presença que atende, de forma cativa, uma parte significativa (mas declinante) do mercado.

Isto posto, a consequência é que sobra muito pouco espaço (mercado, recursos financeiros, ímpeto, etc...) para novas arquiteturas que requeiram mudanças muito radicais nas "bases", como é o caso das Arquiteturas de Fluxo de Dados.

A despeito destes fatos, quando as barreiras da física se tornarem fortes empecilhos para o aumento da velocidade de chaveamento dos componentes e se não aparecer nenhum fato novo na fotônica digital ou na lógica multivariada, é possível que, então, Fluxo de Dados tenha a sua chance.

Este conceito decorre da seguinte constatação: se considerarmos um processo qualquer, o qual se supõe divisível em várias etapas, sendo algumas dessas etapas dependentes de outras; não pode haver nada melhor do que entregar diretamente, tão logo que possível, de forma totalmente assíncrona e com controle totalmente descentralizado, aquilo que uma etapa subsequente precisa para iniciar ou dar andamento no seu subprocesso. Este é o paradigma da Arquitetura Fluxo de Dados.

Em tempo, é possível até que a chance mencionada acima se materialize, mais cedo do que se supõe, por caminhos inversos. Os microprocessadores RISC superescalares atuais já possuem da ordem de 10 unidades funcionais. Na medida em que este número aumentar, um esquema mais elaborado do que simples "unidades de reserva" terá que ser desenvolvido para que as unidades funcionais possam ser plenamente aproveitadas.

Estrutura do artigo: Na seção 2, é descrito o modelo de computação paralela no qual o trabalho é baseado. A seção 3 aborda a exploração de paralelismo em estruturas iterativas. Na seção 4 é apresentada a arquitetura onde programas descritos no modelo poderão executar com alto grau de paralelismo. A seção 5 apresenta as conclusões do trabalho.

Finalmente, é importante notar que as descrições que se pretende fazer aqui, tanto do modelo, como dos esquemas para exploração de paralelismo, estão mais para uma "exposição de idéias" e menos para "demonstração formal de conceitos", como convem ao escopo do artigo.

2. O Modelo

O modelo (que já foi descrito antes, com pequenas modificações, em [7]) é baseado em um grafo direcionado cujos nós têm uma ou duas entradas e uma única saída. Os nós são interligados por arcos por onde trafegam as informações que são processadas nos nós, isto é, os arcos representam dependências de dados e caminhos para esses dados. Notar que este conceito é radicalmente diferente daquele no qual os trabalhos desenvolvidos no MIT [1] e em Manchester [2] se basearam, onde os arcos representam somente dependências de dados.

Os dados que chegam na entrada de um nó são enfileirados, seguindo a ordem de chegada, e vão sendo consumidos pelo nó, na medida em que chegam os dados da outra entrada, se for o caso, e também dentro da própria capacidade de processamento do nó, que é considerada finita. A maioria dos nós de duas entradas requer que haja uma ficha e um dado em cada entrada para que seja iniciada a operação correspondente ao nó ("disparo" do nó), a qual toma um tempo finito e que irá gerar como resultado, via de regra, uma ficha e um dado, que serão enviados a todos os nós descendentes, pelos arcos que os interligam. As entradas dos nós têm filas tipo "primeiro a entrar/primeiro a sair" de capacidade finita. Pouco antes de cada fila de entrada encher, é enviada para o nó ascendente uma [FICHA PARE] que simplesmente irá parar o processamento deste nó (o ascendente) até que o nó descendente que mandou a [FICHA PARE], mande uma [FICHA SIGA], reiniciando o processamento e o envio de fichas e dados. Se um determinado nó receber uma [FICHA PARE] de mais de um descendente, então só deverá reiniciar seu processamento após o recebimento das [FICHA SIGA]'s de cada um dos descendentes que enviaram [FICHA PARE]'s.

Pelos arcos trafegam fichas e dados (lógicos ou aritméticos), que se constituem nas variáveis escalares. No que diz respeito a variáveis indexadas (dados estruturados), só as respectivas fichas circulam no grafo. No momento em que um elemento de um vetor ou matriz for referenciado ou modificado, só então é feito o acesso à memória onde o dado se encontra, através de nós específicos, que irão realizar a leitura ou escrita em uma memória onde estes dados estruturados estarão armazenados. O momento da leitura ou escrita de uma variável indexada depende da chegada da ficha correspondente no nó que realizará a operação.

Os nós serão graficamente representados conforme esta mostrado na figura 1.

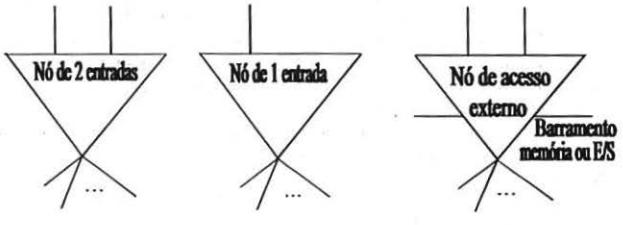

Figura 1 - Representação Gráfica dos Nós

A interligação de nós, segundo regras que serão descritas mais adiante, forma grafos cuja representação gráfica é dada na figura 2,

Figura 2 - Representação Gráfica dos Grafos

onde cada entrada do grafo corresponde à entrada de algum nó do grafo, e cada saída corresponde também à saída de algum nó do grafo.

No que tange à programação, qualquer linguagem estruturada pode ser usada como forma de expressão para posterior tradução para o modelo. As dificuldades são as mesmas que existem na programação de máquinas paralelas, com uma restrição adicional referente ao uso de GO TO's e assemelhados, os quais se constituem em enormes impecilios para a tradução para o modelo.

2.1 Regras Básicas para a Construção de Grafos no Modelo

- 1) Para cada entrada de um nó, vale uma das seguintes afirmações:
  a) está ligada a uma única saída de outro nó (nó ascendente), via um arco;
  b) é uma entrada de um grafo.
- 2) Para a saída de qualquer nó, vale a seguinte afirmação:
  a) está ligada a uma ou mais entradas de outros nós (nós descendentes) via arcos e/ou é uma saída de um grafo.
- 3) Pelos arcos trafegam as seguintes informações na direção saída de um nó, entrada de outro nó:
  a) [FICHA VERDE] ou
  b) [FICHA VERDE] [DADO LÓGICO] ou
  c) [FICHA VERDE] [DADO ARITMÉTICO] ou
  d) [FICHA VERMELHA].
- 4) Na direção contrária, isto é, contra o aparente fluxo normal, trafegam (eventualmente):
  a) [ANTI-FICHA] ou
  b) [FICHA PARE] ou
  c) [FICHA SIGA].
- 5) A entrada de um nó que só consome dados lógicos deve necessariamente estar ligada à saída de um nó que produza dados lógicos. Idem para nós que consomem dados aritméticos. Se uma entrada só consome fichas, então a ligação à saída de qualquer tipo de nó é aceitável. Finalmente, aqueles nós que consomem qualquer tipo de dado podem estar ligados à saída de qualquer nó.
- 6) Considerando a seguinte regra: se o nó A é ascendente de B e B é ascendente de C, segue-se que A é ascendente de C; então, um nó não pode ser ascendente de si mesmo, a não ser por meio de uma construção específica, denominada Grafo Iterativo, que será discutida mais adiante.
- 7) As entradas de um grafo correspondem aos arcos que são caminhos para as variáveis (fichas e dados, se escalares, ou só fichas, se estruturadas) que são referenciadas e/ou modificadas no grafo. As saídas correspondem aos arcos que são caminhos para as variáveis que são referenciadas e/ou modificadas no grafo e que serão modificadas e/ou referenciadas posteriormente, fora do grafo.

  Para facilitar a descrição dos nós, será adotada a seguinte convenção:

  FEE - ficha da entrada esquerda,
  FED - ficha da entrada direita,
  DLEE - dado lógico da entrada esquerda,
  DLED - dado lógico da entrada direita,
  DAEE - dado aritmético da entrada esquerda,
  DAED - dado aritmético da entrada direita,

FS - ficha da saída,
DLS - dado lógico da saída,
DAS - dado aritmético da saída,

e os vários tipos de nós do modelo serão divididos em classes, as quais serão descritas a seguir. Fica convencionado que para os nós que só têm uma entrada, essa entrada será a esquerda. Diremos que uma variável é referenciada se ela estiver do lado direito do sinal de atribuição e modificada se estiver do lado esquerdo do sinal de atribuição do programa fonte correspondente.

2.2 Descrição dos Vários Tipos de Nós

2.2.1 Nós da Classe I

2.2.1.1 Nós de operações aritméticas ou lógicas simples, com duas entradas e uma saída, cujo funcionamento segue o seguinte procedimento:

Havendo, pelo menos, uma ficha em cada fila de entrada e estando o nó em repouso, o mesmo será disparado, isto é, a ficha e o dado das cabeças das filas serão consumidos, a operação (OP) será realizada nos dados, e uma ficha e um dado serão produzidos e uma cópia enviada para todos os nós descendentes, conforme tabela que se segue:

Tabela 1

ENTRADA ESQUERDA FICHA CONSUMIDA	ENTRADA DIREITA FICHA CONSUMIDA	SAÍDA	
		FICHA PRODUZIDA	DADO
verde	verde	verde	(*)
outras combinações de cores das fichas		vermelha	-

(*) Dependendo do tipo do nó, os possíveis dados de saída são: DLS = DLEE (OP) DLED;
DAS = DAEE (OP) DAED;
DLS = DAEE (OP) DAED.

2.2.1.2 Nós de operações aritméticas ou lógicas simples, com uma entrada e uma saída, cuja operação segue o seguinte procedimento:

Havendo pelo menos uma ficha na única fila de entrada e estando o nó em repouso, a ficha e o dado da cabeça da fila são consumidos, a operação (OP) do nó é realizada no dado, e uma ficha e um dado são produzidos e enviados para todos os nós descendentes, conforme tabela que se segue:

Tabela 2

ENTRADA ESQUERDA FICHA CONSUMIDA	SAÍDA	
	FICHA PRODUZIDA	DADO
verde	verde	(**)
vermelha	vermelha	-

(**) Dependendo do tipo do nó, os possíveis dados de saída são: DAS = constante (OP) DAEE;
DLS = (OP) DLEE;
DAS = (OP) DAEE;
DAS = constante;
DLS = constante.

Em ambos os casos, isto é, tanto para nós de uma entrada como nós de duas entradas, se a ficha de saída for vermelha, ela é enviada aos nós descendentes imediatamente, não havendo nenhum processamento ou dado associado.

Exemplos de nós da Classe I:

- Nó de adição: segue a tabela 1 e a saída é DAS = DAEE + DAED
- Nó da operação lógica .OU.: segue a tabela 1 e a saída é DLS = DLEE .OU. DLED
- Nó de comparaçao de dados aritméticos: segue a tabela 1 e a saída é DLS = DAEE (OP) DAED
- Nó de negação lógica: segue a tabela 2 e a saída é DLS = .NOT. DLEE
- Nó para gerar um valor constante: segue a tabela 2 e a saída é DAS ou DLS = constante
- Nó para comparar dado com constante: segue tabela 2 e DLS = DAEE (OP) constante.

2.2.2 Nós da Classe II

Esta classe é composta por nós que têm a função de controle do fluxo de dados. Cada tipo de nó desta classe será descrito individualmente.

- Nó tipo **PV** (passe se verdadeiro): Este nó passa a ficha e o dado que esta na cabeça da fila da entrada esquerda para a saída se a ficha da entrada da direita for verde e o dado lógico da entrada da direita (DLED) for verdadeiro. Em todos os outros casos, a saída será uma ficha vermelha. Se na entrada da esquerda só ocorrer uma ficha, isso será indicado na tabela com o símbolo ϕ.

ENTRADA ESQUERDA	ENTRADA DIREITA		SAÍDA	
FICHA CONSUMIDA	FICHA CONS.	DLED	FICHA PRODUZIDA	DADO
verde	verde	verdadeiro	verde	ϕ, DAEE,DLEE
verde	verde	falso	vermelha	-
outras combinações			vermelha	-

- Nó tipo **PF** (passe se falso): Este nó é muito parecido com o nó tipo **PV**, com a diferença que o dado lógico da entrada direita (DLED) deve ser falso para que o dado da esquerda passe para a saída.
- Nó tipo **XOR** (passe um ou o outro exclusivamente): Este nó passa para a saída o dado que estiver acompanhado de uma ficha verde. Se ambas entradas tiverem fichas verdes, a saída será uma ficha vermelha, conforme tabela a seguir:

ENTRADA ESQUERDA	ENTRADA DIREITA	SAÍDA	
FICHA CONSUMIDA	FICHA CONSUMIDA	FICHA PRODUZIDA	DADO
verde	verde	vermelha	-
verde	vermelha	verde	ϕ, DAEE, DLEE
vermelha	verde	verde	ϕ, DAED, DLED
vermelha	vermelha	vermelha	-

Os tres nós descritos acima (**PV, PF e XOR**), em conjunto, permitirão a construção de estruturas condicionais do tipo *if (GRAFO1) then GRAFO2*. Mais adiante, o protótipo do grafo condicional será descrito detalhadamente.

- Nó tipo **IT** (iteração): Este nó é usado na implementação de estruturas iterativas. Seu funcionamento difere dos demais no sentido de que possue 2 estados estáveis, que serão chamados de inicial (IN) e realimentado (REL). Outra diferença é que este nó, apesar de ter duas entradas, é disparado com fichas de apenas uma entrada. Seu funcionamento é descrito a seguir:

ENTRADA ESQUERDA FICHA CONSUMIDA	ENTRADA DIREITA FICHA CONSUMIDA	ESTADO	SAÍDA		
			FICHA PRODUZIDA	DADO	NOVO ESTADO
verde	não consumida	INI	verde	ϕ, DAEE ou DLEE	REL
vermelha	não consumida	INI	vermelha	-	REL
não consumida	verde	REL	verde	ϕ,DAED ou DLED	REL
não consumida	vermelha	REL	não produzida	-	INI

- Nó tipo **FIT** (fim de iteração): Este nó, em conjunto com o nó tipo **IT**, formam a base para a implementação de construções iterativas. Seu funcionamento é descrito a seguir, através da correspondente tabela:

ENTRADA ESQUERDA	ENTRADA DIREITA	SAÍDA	
FICHA CONSUMIDA	FICHA CONSUMIDA	FICHA PRODUZIDA	DADO
verde	verde e se DLED=falso	verde	ϕ, DAEE, DLEE
verde	verde e se DLED=verdadeiro	não produzida	-
verde	vermelha	vermelha	-
vermelha	verde	vermelha	-
vermelha	vermelha	vermelha	-

Para cada variável lógica, aritmética ou indexada que for referenciada e/ou modificada no corpo da estrutura iterativa, haverá um par de nós **IT** e **PV**. Se esta variável for referenciada ou modificada mais adiante, também haverá um nó tipo **FIT** que permitirá a "saída" do dado para fora da estrutura iterativa. Mais adiante, o protótipo do grafo iterativo será descrito em detalhe.

- Nó tipo **INI** (início): Este nó não tem entradas e gera uma única ficha verde, quando recebe um comando para tal de algum supervisor. O funcionamento é trivial.
- Nó tipo **FIM:** Este nó tem duas entradas e não tem saída. A tabela que o descreve se segue:

ENTRADA ESQUERDA FICHA CONSUMIDA	ENTRADA DIREITA FICHA CONSUMIDA	
verde	verde	envia sinal para supervisor indicando que a computação foi bem concluida
outras combinações		envia sinal para supervisor indicando que a computação não foi bem concluida

2.2.3 Nós da Classe III

Esta classe é composta pelos nós que irão propiciar o acesso a memórias, o que permitirá a computação com dados estruturados, semáforos etc.... Nesta classe, também estão os nós que permitirão operações de entrada/saída.

- Nó tipo **LM** (ler memória): Este nó realiza a operação de leitura da memória de dados estruturados. A entrada da esquerda (DAEE) corresponde ao endereço de memória ao qual se deseja fazer o acesso. A entrada da direita (da qual só interessa a ficha) é usada para sincronizar com outras operações de acesso à memória. Este acesso é feito por um barramento ao qual estão também ligados todos os outros nós que fazem leitura e escrita na memória. A tabela que descreve o funcionamento é dada a seguir:

ENTRADA ESQUERDA FICHA CONSUMIDA	ENTRADA DIREITA FICHA CONSUMIDA	SAÍDA	
		FICHA PRODUZIDA	DADO
verde	verde	verde	dado lido da memória na posição DAEE
outras combinações		vermelha	-

- Nó tipo **EM** (escrever na memória): Este nó realiza a operação de escrita na memória. A entrada da esquerda (DAEE) corresponde ao endereço e a entrada da direita (DAED) é o dado a ser armazenado. A ficha de saída só será enviada para os nós descendentes quando a operação tiver sido realizada e serve para sincronizar os próximos acessos à mesma variável indexada. A tabela é conforme se segue:

ENTRADA ESQUERDA FICHA CONSUMIDA	ENTRADA DIREITA FICHA CONSUMIDA	SAÍDA FICHA PRODUZIDA
verde	verde	verde
outras combinações		vermelha

- Nó Tipo **ESP** (espera): Este nó, junto com o nó tipo **AVIsa**, permite a implementação de regiões críticas. Na entrada da esquerda do nó ESP, só é relevante a cor da ficha. Na entrada da direita, o dado (DAED) será o número do semáforo. A ficha da saída só será produzida quando o semáforo identificado por DAED permitir, isto é, o seu valor for maior ou igual a zero, e neste caso ele será decrementado de um. A operação que testa e decrementa o semáforo deve ser considerada indivisível. A tabela que descreve o funcionamento é idêntica à do nó **EM**.
- Nó tipo **AVI** (avisa): O nó tipo **AVIsa** soma um ao semáforo identificado pelo dado DAED da direita. Na entrada da esquerda, só é relevante a cor da ficha. A tabela que descreve o funcionamento é idêntica à tabela do nó tipo **EM**.
- Nó tipo **SINC** (sincroniza): Este nó simplesmente passa para a saída, a entrada da esquerda se a ficha da direita for verde. Se a ficha da direita for vermelha, a saída será uma ficha vermelha. A utilidade deste nó é para compor sincronismos de acessos às memórias (variáveis indexadas e semáforos). A tabela se segue:

ENTRADA ESQUERDA FICHA CONSUMIDA	ENTRADA DIREITA FICHA CONSUMIDA	SAÍDA	
		FICHA PRODUZIDA	DADO
verde	verde	verde	ϕ, DAEE ou DLEE
outras combinações		vermelha	-

- Nó tipo **.E.F** (.E. FICHA): Este nó tem a finalidade de compor as últimas operações nos dados de um grafo para permitir a detecção do final de uma computação. A tabela que descreve o seu funcionamento é idêntica à do nó **EM**.
- Nó tipo **SA** (saída): O nó **SA**ída tem por objetivo implementar a operação de saída de dados em um dispositivo de E/S identificado pela porta de entrada da esquerda. O dado enviado é o da porta da direita. Quando a operação tiver sido concluída com sucesso, será produzida uma ficha verde que poderá ser usada para sincronizar outras operações na mesma cadeia de saída. A tabela que descreve o seu funcionamento é idêntica à do nó **EM**.
- Nó tipo **EN** (entrada): O nó **EN**trada tem por objetivo implementar a operação de entrada de dados de um dispositivo de E/S identificado pela porta de entrada da esquerda. Da porta da direita só interessa a ficha, que pode ser usada para sincronismo. O dado lido do dispositivo vai para a porta de saída do nó. A tabela que descreve o funcionamento é dada seguir:

ENTRADA ESQUERDA FICHA CONSUMIDA	ENTRADA DIREITA FICHA CONSUMIDA	SAÍDA	
		FICHA PRODUZIDA	DADO
verde	verde	verde	dado lido do "stream"
outras combinações		vermelha	-

2.3 Grafos Bem Construídos e Computações

No decorrer desta seção, serão definidas quatro regras para construção de grafos, os quais serão denominados "grafos bem construidos".

Regra 1: Todo grafo formado por nós da Classe I e mais nós dos tipos **LM** e **.E.F**, e que seja construído segundo as regras descritas na seção 2.1, é um grafo bem construído.

Regra 2: A composição de grafos bem construídos resulta em um grafo bem construído.

2.3.1 Protótipo do Grafo Condicional

O Grafo Condicional representa uma construção cujo correspondente em uma linguagem de alto nível (C, por exemplo), é o seguinte: *if (GRAFO1) GRAFO2*; onde GRAFO1 é um grafo bem construído, cuja única saída é do tipo [FICHA] [DADO LÓGICO]. Se o valor desse dado lógico for verdadeiro, então o GRAFO2 (também bem construído) será "executado". Na figura 3, é mostrado o protótipo do Grafo Condicional,

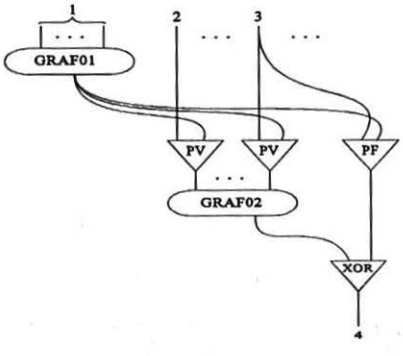

Figura 3 - Protótipo do Grafo Condicional

onde:

- 1. são os arcos que transportam fichas e dados lógicos ou aritméticos ou fichas de variáveis indexadas que, em todos os casos, são somente referenciadas no GRAFO1;

- 2. representa o arco que transporta a ficha de uma variável lógica ou aritmética escalar ou ainda só a ficha de uma variável indexada que, em todos os casos, pode ser referenciada e/ou modificada no GRAFO2, mas não é referenciada posteriormente;
- 3. representa o arco que transporta a ficha e uma variável lógica ou aritmética escalar ou ainda só a ficha de uma variável indexada que, em todos os casos, pode ser referenciada no GRAFO2, é modificada no GRAFO2 e é referenciada posteriormente, via arco 4.

Segue-se um exemplo de um grafo condicional correspondente ao seguinte: *if (A < B) C = C*D;*

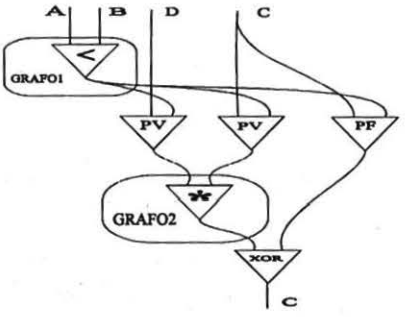

Figura 4 - Exemplo de um Grafo Condicional

Regra 3: Um grafo construído segundo o protótipo do Grafo Condicional é um grafo bem construído.

2.3.2 Protótipo do Grafo Iterativo

O Grafo Iterativo representa uma construção, cujo correspondente em uma linguagem de alto nível (C, por exemplo), é o seguinte: *while (GRAFO1) GRAFO2;* onde GRAFO1 é um grafo bem construído que só tem uma saída do tipo [FICHA] [DADO LÓGICO]. Enquanto o valor desse dado lógico for verdadeiro, o GRAFO2 será executado iterativamente. Na figura 3, é mostrado o protótipo do Grafo Iterativo.

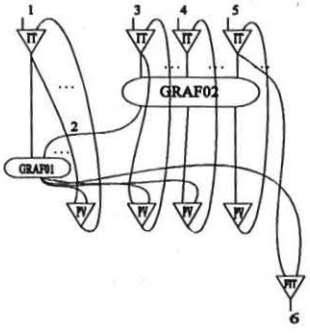

Figura 5 - Protótipo do Grafo Iterativo

onde:

- 1. representa o arco que transporta a ficha e uma variável lógica ou aritmética escalar ou ainda a ficha de uma variável indexada que é referenciada no GRAFO1;
- 2. representa o arco que transporta a ficha de uma variável local lógica ou aritmética escalar que é referenciada no GRAFO1;

- 3. representa o arco que transporta a ficha e uma variável lógica ou aritmética escalar ou a ficha de uma variável indexada que é referenciada no GRAFO2;
- 4. representa o arco que transporta a ficha e uma variável lógica ou aritmética escalar ou a ficha de uma variável indexada que é referenciada e/ou modificada no GRAFO2 e que não será mais referenciada ou modificada posteriormente, fora da construção iterativa;
- 5. representa o arco que transporta a ficha e uma variável lógica ou aritmética escalar ou a ficha de uma variável indexada que é referenciada e/ou modificada no GRAFO2 e que será referenciada e/ou modificada posteriomente, fora da construção iterativa;
- 6. representa o arco que transporta a ficha e a variável (ou só a ficha) correspondente ao arco (5) para outros grafos "descendentes".

Regra 4: Um grafo construído segundo o protótipo do Grafo Iterativo é um grafo bem construído.

Tendo em vista o aspecto computabilidade, as quatro regras descritas anteriormente permitem afirmar que o presente modelo é computacionalmente completo, ou seja, qualquer função computável pode ser representada no modelo. Como de praxe, deve-se supor que os nós e arcos podem operar com números tão grandes quanto se desejar. Outras construções mais elaboradas podem analogamente (e facilmente) ser implementadas no modelo, assim como foram implementadas as construções condicional e iterativa.

2.3.3 A Computação

Uma computação é um grafo que tem um único nó **INÍcio** e um único nó **FIM**, conforme desenho a seguir:

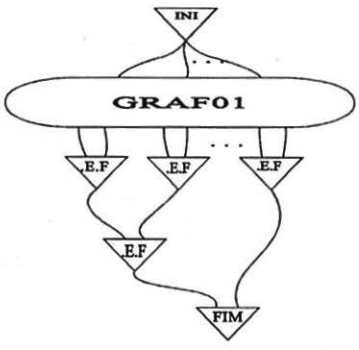

Figura 6 - Uma Computação

onde o GRAFO1 é um grafo bem construído cujas saídas estão ligadas a nós do tipo **.E.F** de forma que a necessária presença de uma ficha em cada saída do GRAFO1 possa ser detectada pelo nó **FIM**. A importância da definição formal de uma computação, conforme descrição acima, se prende à necessidade da detecção do fim da computação, com a finalidade de desalocar (disponibilizar) nós na implementação da máquina baseada no modelo, como veremos adiante.

Uma computação, conforme definição acima, é sempre deterministica. As questões relacionadas com determinismo são as mesmas que ocorrem no âmbito da multiprogramação ou multiprocessamento onde há necessidade de sincronismo nos acessos (leituras e escritas) às memórias que, no modelo, armazenam os dados estruturados ou semáforos. Desta forma, é perfeitamente possível garantir o determinismo de computações que possuam nós de escrita na memória bem como de acessos a semáforos, desde que leituras e escritas sejam devidamente sequenciados, assim como devem ser sequenciados os acessos a memórias em ambientes com multiprogramação ou multiprocessamento.

2.3.4 Exemplo de uma Computação

A seguir é mostrado um programa que corresponde a um código iterativo para cálculo do fatorial de N:

read(N);
FATORIAL = 1;

```
while (N > 1) { FATORIAL = FATORIAL * N;
                N = N - 1;
              }
print(FATORIAL); .
```

O grafo (computação) correspondente é o seguinte:

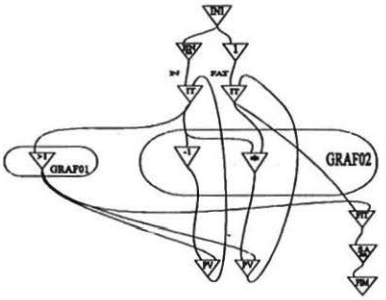

Figura 7 - Exemplo de uma Computação.

3. Exploração de Paralelismo em Estruturas Iterativas

Além do paralelismo que brota é explorado naturalmente, o modelo permite reestruturações nos grafos, que podem ser facilmente identificadas e implementadas, as quais irão propiciar um grau adicional de paralelismo. Exemplos triviais são aqueles que aproveitam as propriedades de associatividade, distributividade e comutatividade nas operações aritméticas simples. Quanto ao paralelismo em construções iterativas, existem possibilidades que são particularmente interessantes e que serão discutidas a seguir.

De uma forma muito simplista, podemos classificar os procedimentos que implementam algorítmos em uma determinada linguagem, conforme a seguinte taxonomia:

a) Absolutamente lineares, isto é, que não possuem construções iterativas.

b) Iterativos, sem dependências de dados entre iterações. Exemplo:

```
DO I = 1,N
A(I) = (B(I) + C(I)) * D(I)
ENDDO
```

c) Iterativos, com dependências de dados entre iterações, dependências essas que não existiam no algoritmo original. Exemplo:

```
DO I = 1,N
T = B(I) + C(I)
A(I) = T * D(I)
ENDDO
```

d) Iterativos, com dependências de dados entre iterações, sendo essas dependências intrínsecas ao algoritmo. Exemplo:

```
DO I = 2,N
A(I) = A(I-1) + A(I+1)
ENDDO
```

A pesquisa atual na área de compiladores [8] permite supor que boa parte do paralelismo existente em a), b) e c) já é ou será explorado brevemente nas máquinas paralelas modernas. Existe, todavia, uma quantidade considerável de paralelismo que é aquele que se enquadra no item d), cuja exploração é menos considerada nos compiladores modernos. Este tipo de paralelismo é objeto desta seção. Mais especificamente, o que se pretende é explorar paralelismo através da reestruturação do grafo resultante do "desenrolamento parcial" de uma construção iterativa. Considerando o exemplo da seção 2.3.4, se procedermos o desenrolamento dessa construção de modo a reduzir o número de iterações por um fator 3, teremos o grafo A da figura 8,

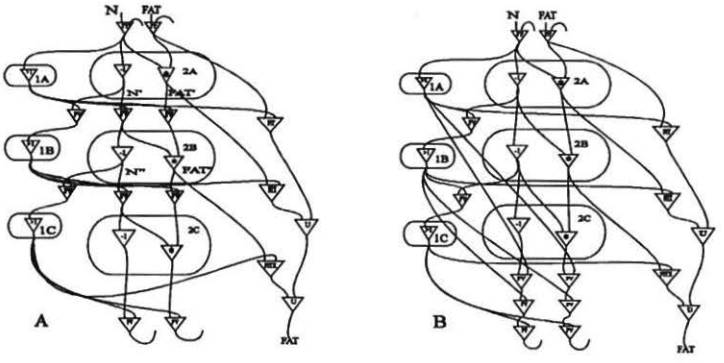

Figura 8 - Desenrolamento de uma Construção Iterativa

onde os grafos 2A, 2B e 2C são idênticos ao GRAFO2 da figura 7, e o nó **UNI**ão é um novo tipo de nó que simplesmente passa para a saída qualquer [FICHA] [DADO] que chegar a qualquer das suas duas entradas, obedecendo a ordem de chegada. O nó **FITX** também é um novo tipo de nó cujo funcionamento é parecido com o do nó **FIT** e deve ser usado no lugar do nó **FIT** quando se proceder o desenrolamento de construções iterativas, da segunda cópia do grafo em diante, isto é, a primeira cópia do GRAFO2 terá um nó tipo **FIT**, as subsequentes nós tipo **FITX**. A tabela do nó **FITX** se segue:

♦ Nó tipo **FITX** (fim de iteração no desenrolamento de construções iterativas):

ENTRADA ESQUERDA FICHA CONSUMIDA	ENTRADA DIREITA FICHA CONSUMIDA	SAÍDA	
		FICHA PRODUZIDA	DADO
verde	verde e se DLED=falso	verde	ϕ, DAEE ou DLEE
verde	verde e se DLED=verdadeiro	não produzida	-
verde	vermelha	não produzida	-
vermelha	verde	não produzida	-
vermelha	vermelha	não produzida	-

Os nós do tipo **PV** que estão entre os grafos 2A e 2B da figura 8A funcionam como "validadores" dos dados que saem do grafo 2A para o grafo 2B. Esta função pode muito bem ser exercida na saída do grafo 2B, o que permitirá a ligação direta entre os grafos 2A e 2B, conforme mostrado na figura 8B. Idem para o grafo 2C. É aí que reside a fonte de melhoria do tempo de processamento, via a reestruturação do grafo resultante da composição de 2A, 2B e 2C da figura 8B, sendo o respectivo resultado mostrado na figura 9. No caso, a reestruturação que foi realizada, simplesmente aproveita a propriedade da associatividade da multiplicação e transforma ((N*FAT)*(N-1))*(N-1-1) em (N*FAT)*((N-1)*(N-2)), propiciando, com isso, o ganho do tempo de uma multiplicação menos o tempo de uma subtração, a cada 3 iterações. Notar que os sub-grafos que produzem os resultados intermediários N', N'', FAT', e FAT'' devem ser preservados isoladamente, com os respectivos nós tipo **PV**.

Extendendo este procedimento, isto é, procedendo o desenrolamento de um número maior de iterações, pode-se explorar paralelismo de uma forma bastante significativa. Notar também que o processo é perfeitamente sistematizável.

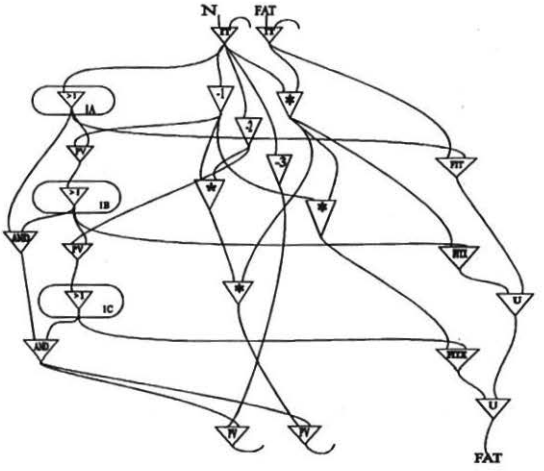

Figura 9 - Composição de 3 Iterações.

A seguir, será mostrado um exemplo onde há acessos a dados estruturados. Suponhamos a construção iterativa a seguir:

```
DO I = 1, M
Y(I) = Y(I) + X(I)
ENDDO
...
... = ...Y(K)...
```

O grafo correspondente é o seguinte:

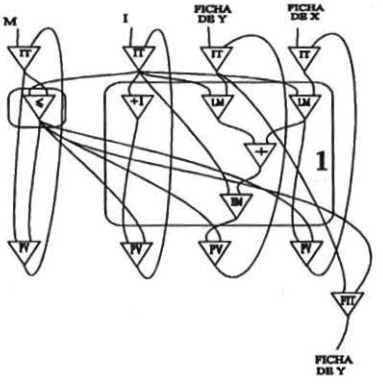

Figura 10 - Grafo de uma Construção Iterativa com Acessos a Dados Estruturados

Como não existem dependências de dados entre as iterações deste "loop", ele é facilmente desenrolável. Se M for conhecido em tempo de compilação, então pode-se simplesmente "copiar" M vezes o grafo 1 de forma a executar as M iterações em paralelo, cada uma com o valor apropriado de I. Se M não for conhecido em tempo de compilação, ainda assim é perfeitamente possível desenrolar um

número julgado apropriado de iterações, de forma que o número final de iterações seja bastante reduzido. Por exemplo, os cálculos de Y(I) = Y(I) + X(I) para I, I+1, I+2, ... I+9 poderiam ser realizados em paralelo, de forma que o número de iterações seja reduzido por um fator de 10.

A reestruturação sugerida acima é dificultada quando a dependência entre iterações for de dados estruturados. Neste caso, há que se cuidar para que leituras e escritas na mesma posição de memória guardem a ordem originalmente desejada de execução. Em adição, quando se proceder o deslocamento de nós tipo PV, não é possível "saltar" um nó de escrita na memória. Neste caso, deve-se preservar inalteradas as partes do grafo que compõe as operações que escrevem na memória e, se possível, fazer referência aos dados que causam dependência antes da escrita na memória.

3.1 A Anti-Ficha

O objetivo da anti-ficha é permitir a não execução de um grafo, ou parte dele, do qual se sabe, antecipadamente, que o resultado não será aproveitado. A anti-ficha, como foi dito anteriormente, trafega na direção contrária às fichas. As regras de funcionamento da anti-ficha são as seguintes:

- 1. Quando uma anti-ficha encontra uma ficha em um arco, ambas se cancelam.
- 2. Uma anti-ficha só pode entrar em um nó (pela saída), se todos os arcos de saída deste nó já estiverem com, pelo menos, uma anti-ficha na espera, e aí, de todos os arcos de saída, será consumida uma anti-ficha.
- 3. Se o nó que recebeu uma anti-ficha estiver ocupado realizando algum processamento, este processamento é cessado e não são enviadas fichas para os nós descendentes.
- 4. se o nó estiver em repouso, então a anti-ficha continua seu trajeto, de "subida" por ambas entradas (se houverem), até encontrar uma ficha e ambas (ficha e anti-ficha) se cancelam.
- 5. Quando uma anti-ficha encontra um nó tipo **IT**, **FIT**, **UNI**, **SA** ou **EM**, então ela pára, e aí fica aguardando a "sua" ficha.
- 6. Uma anti-ficha é gerada quando em um nó do tipo **PV**, a entrada da direita for falsa (ou em um nó **PF**, a entrada da direita for verdadeira), e a ficha da entrada esquerda não tiver chegado, sendo então gerada uma ficha vermelha para os nós descendentes e uma anti-ficha, que "subirá" pela entrada da esquerda.

A seguir, será mostrado um exemplo do "funcionamento" da anti-ficha. Suponhamos o seguinte trecho de programa, que poderia estar no interior de uma construção iterativa:

```
C = X**2 + X**3;
if(A=B) C = D;
```

cujo grafo correspondente é:

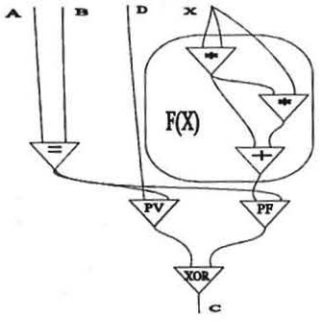

Figura 11 - Exemplo de Aplicação da Anti-Ficha.

Se *A=B* para uma determinada iteração, então não há necessidade do cálculo de *C = F(X) = X**2 + X**3* ser concluído, podendo o grafo *F(X)* ser ocupado com a próxima iteração (se o próximo *X* já estiver disponível), economizando-se assim o tempo do "resto" do cálculo de *F(X)*. No caso, a anti-ficha será gerada na entrada esquerda do nó **PF** e "subirá" pelo grafo *F(X)*, autocancelando-se com as fichas que for encontrando. A rigor, mesmo que *F(X)* seja um grafo iterativo (bem construído), é possível definir um procedimento (que não será descrito aqui) para aproveitar a "subida" de anti-fichas. Os obstáculos definitivos para a anti-ficha são os nós que produzem efeitos colaterais (**EM** e **SA**).

3.2 Término de uma Computação

Uma computação termina quando o nó **FIM** recebe a sua única ficha, a qual deverá ser verde para indicar que a computação foi bem sucedida, e não houver mais nenhuma ficha ou anti-ficha no grafo.

4. A Arquitetura

A arquitetura ideal, que ultrapassa em muito os limites atuais de viabilidade prática de realização, é aquela em que o modelo é diretamente mapeado em uma estrutura que possua unidades funcionais que executem as operações de cada tipo de nó, interligadas por uma chave "crossbar", onde cada entrada de nó pode ser ligada diretamente à saída de qualquer outro nó, conforme a figura 12, e onde o número de unidades funcionais é maior do que o número de nós do maior grafo que se deseja executar.

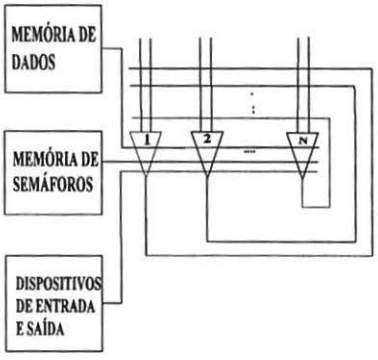

Figura 12 - A Arquitetura Ideal.

Carregar um programa nesta máquina, significa atribuir a cada unidade funcional a tarefa de um nó do grafo e estabelecer as ligações das chaves de forma a mapear o grafo na máquina.

Passando para o mundo real, e dentro do possível não abandonando os seguintes princípios norteadores:

- descentralizar o controle, onde possível;
- usar métodos assíncronos, onde possível;
- aceitar um mapeamento de nós para unidades funcionais do tipo "muitos para um";
- procurar uma solução que ofereça caminhos diretos, para que operandos possam rapidamente passar de onde são produzidos para onde são consumidos;
- evitar unidades centrais de alocação de recursos ou arbitramento;

e considerando a construção iterativa como algo a ser particularmente privilegiado, chega-se a uma configuração básica de disposição das unidades funcionais e respectivas interligações, que se assemelha à superfície de um cilíndro. "Planificando" este cilíndro, a aparência seria conforme esta mostrado na figura 13, onde M1 é uma memória de dados estruturados ligada, via um barramento, a uma coluna de unidades funcionais, SEM é uma memória para semáforos, ligada a outra coluna de unidades funcionais e E/S são dispositivos de entrada/saída, ligados a outra coluna de unidades funcionais.

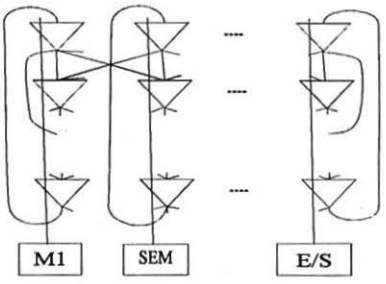

Figura 13 - Unidades Funcionais e Interligações.

Vendo a figura 13 como uma matriz, a saída de qualquer unidade funcional se liga diretamente às entradas de algumas unidades da linha seguinte e a algumas unidades da mesma coluna. Para uma máquina com 24 unidades funcionais e uma capacidade de multiplexação de 1 para 8 nas saídas e entradas das unidades funcionais, uma configuração possível seria a da figura 14,

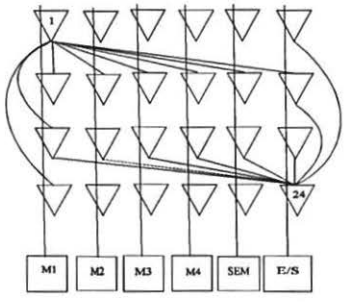

Figura 14 - Possível Configuração.

onde só estão mostradas as ligações da unidade funcional 1 com as suas descendentes, e da unidade 24 com as suas ascendentes. Neste caso, qualquer unidade funcional pode mandar dados para qualquer outra unidade com, no máximo, um passo intermediário. Um novo tipo de nó (trivial) de uma entrada se faz necessário para atuar como nó de passo intermediario. Na figura 14, M1, M2...são bancos de memória para dados estruturados, SEM é uma memória de semáforos e E/S são unidades de entrada/ saída.

Outras configurações com a mesma capacidade de multiplaxação, e os passos intermediários limitados a um, são perfeitamente possíveis. Se admitirmos dois passos intermediários, o número de unidades funcionais da máquina pode crescer significativamente (mantendo-se a multiplexação de 1 para 8).

A implementação física destas configurações é viável se a capacidade de multiplexação e demultipexação das entradas e saídas de 1 para 8 for integrável. Se cada porta tiver aproximadamente 70 bits de largura (32 bits para dados, 20 para o número do nó e 18 para controles, por exemplo), isso implica em dispositivos LSI de aproximadamente 630 pinos (9 x 70). Pressupõe-se que é importante construir o multiplexador e o demultiplexador em dispositivos LSI monolíticos. Notar que com 20 bits para o número do nó, em cada unidade funcional poderão ser mapeados até um milhão de nós (2**20).

4.1 A Unidade Funcional

Cada nó de um grafo deverá, de forma estática, ser mapeado em uma unidade funcional, o que implica que junto com as fichas e dados, também devem ser enviados os números do nó a que se destinam as informações na unidade funcional, bem como a informação que indica a qual das entradas

(esquerda ou direita) o dado se destina. Desta forma, os possíveis "pacotes" de informação que trafegam da saída de uma unidade funcional para a entrada de outra unidade funcional, são:

PACOTE TIPO	NUM. DO NÓ NA UNID. DESC.	ENTRADA	FICHA	DADO
1	NN bits	esquerda/direita	verde/verm.	-
2	NN bits	esquerda/direita	verde	falso/verdadeiro
3	NN bits	esquerda/direita	verde	dado aritmético 1
4	NN bits	esquerda/direita	verde	dado aritmético 2
...				
P	NN bits	esquerda/direita	verde	dado aritmético k

onde os pacotes 3, 4...P cobrem os vários tipos de dados aritméticos, inclusive aqueles que devem ser multiplexados no tempo, por terem largura superior à largura da via (arco) e NN é o número de bits para a identificação do nó, permitindo um mapeamento de até 2**NN nós por unidade funcional.

Trafegando em sentido contrário, existem os seguintes tipos de pacotes:

PACOTE TIPO	NÚM. DO NÓ NA UNIDADE ASCENDENTE
P+1 (PARE)	NN bits
P+2 (SIGA)	NN bits
P+3 (ANTI-FICHA)	NN bits

Internamente, cada unidade funcional, com capacidade de mapear 2**NN nós, tem a configuração mostrada na figura 15,

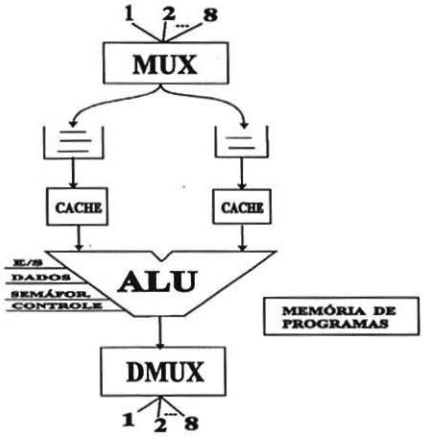

Figura 15 - Unidade Funcional

onde a memória de programa, endereçada pelo número do nó, tem uma configuração de palavra com os seguintes campos (assumindo que o número máximo de descendentes diretos que cada nó pode ter é D):

- TIPO DO NÓ (tantos bits quanto necessários para codificar todos os tipos de nós)
- ESTADO DO NÓ IT (INI ou REL) (1 bit)
- CONSTANTE (64 bits);

Informações relacionadas ao nó descendente 1:

- NÚMERO DA PORTA DE SAÍDA DO DMUX (4 bits);
- NÚMERO DO NÓ NA UNIDADE DESCENDENTE (NN bits)
- INDICAÇÃO DA ENTRADA NO NÓ DESCENDENTE (esquerda ou direita) (1 bit)
- ANTI-FICHAS RECEBIDAS DO NÓ DESCENDENTE (número de bits por determinar)
- RECEBEU UMA FICHA PARE DO NÓ DESCENDENTE (sim/não) (1 bit)

Informações relacionadas ao nó descendente 2:
- NÚMERO DA PORTA DE SAÍDA DO DMUX (4 bits);
- NÚMERO DO NÓ NA UNIDADE DESCENDENTE (NN bits)
- INDICAÇÃO DA ENTRADA NO NÓ DESCENDENTE (esquerda ou direita) (1 bit)
- ANTI-FICHAS RECEBIDAS DO NÓ DESCENDENTE (número de bits por determinar)
- RECEBEU UMA FICHA PARE DO NÓ DESCENDENTE (sim/não) (1 bit)
- ...

Informações relacionadas ao descendente D:
- NÚMERO DA PORTA DE SAÍDA DO DMUX (4 bits);
- NÚMERO DO NÓ NA UNIDADE DESCENDENTE (NN bits)
- INDICAÇÃO DA ENTRADA NO NÓ DESCENDENTE (esquerda ou direita) (1 bit)
- ANTI-FICHAS RECEBIDAS DO NÓ DESCENDENTE (? bits)
- RECEBEU UMA FICHA PARE DO NÓ DESCENDENTE (sim/não) (1 bit)

Entrada esquerda:
- NÚMERO DA PORTA DE ENTRADA DO MUX (4 bits)
- NÚMERO DO NÓ NA UNIDADE ASCENDENTE (NN bits)
- MANDOU UMA FICHA PARE PARA O NÓ ASCENDENTE (sim/não) (1 bit)

Entrada direita:
- NÚMERO DA PORTA DE ENTRADA DO MUX (4 bits)
- NÚMERO DO NÓ NA UNIDADE ASCENDENTE (NN bits)
- MANDOU UMA FICHA PARE PARA O NÓ ASCENDENTE (sim/não) (1 bit)

Na indicação do nó descendente, NÚMERO DA PORTA DE SAÍDA DO DMUX = zero significa que o nó descendente esta na mesma unidade funcional, daí a necessidade de 4 bits para codificar todas as possibilidades.

As memórias CACHE em cada entrada da ALU, também endereçadas pelo número do nó (o que implica que tenham 2**NN palavras), têm a seguinte configuração da palavra:

TIPO	FICHA	DADO
tipo do pacote	verde/vermelha (1 bit)	32 bits

As memórias FILA DE ENTRADA, entre os CACHE's e o MUX, têm os mesmos campos dos CACHE's e mais o número do nó. Notar que agora, quando uma FILA DE ENTRADA estiver prestes a encher, [FICHA PARE]'s devem ser enviadas a todos os nós ascendentes daquela unidade, daquela entrada e cuja posição no CACHE já esteja ocupada.

4.2 Fluxo de Dados na Unidade Funcional

Inicialmente, são carregadas na memória de programas de cada unidade funcional, endereçadas pelo número do nó nesta unidade, as palavras (instruções) que definem os nós pertencentes à computação X que serão mapeados nesta unidade. Após o início da computação X, a unidade deverá receber pacotes das unidades ascendentes, os quais serão colocados na fila apropriada (esquerda ou direita). Os primeiros pacotes de cada nó mapeado irão direto para os respectivos CACHES's.

Notar que nada impede que outra computação esteja executando simultamente, e neste caso teríamos, em uma determinada unidade, o mapeamento dos nós 1 a m, por exemplo, para a computação X e m+1 a m+k para a computação Y, e assim por diante, só limitado pelo número total de nós que podem ser mapeados em cada unidade funcional. Quando a computação X concluir, os nós que ela ocupava na unidade funcional são simplesmente colocados a disposição para outras computações, como um recurso qualquer de um computador multiprogramado. Notar também que a fragmentação de nós não é problema, isto é, depois de algum tempo de uso multiprogramado, a alocação da numeração dos nós em uma determinada unidade funcional pode ficar totalmente fragmentada, sem que isto represente qualquer inconveniente.

Na medida em que as palavras dos CACHE's (pacotes dos vários nós) forem sendo consumidas pela unidade funcional, novos pacotes, se houverem, devem ser retirados das filas e colocados nas respectivas posições dos CACHES's. Desta forma, a despeito do tamanho das filas ser finito, exigindo um esquema para sustar o processamento do nó ascendente ([FICHA PARE]), não haverá "deadlock", pois sempre há espaço garantido nos CACHES's para um conjunto de operandos de todos os nós.

É importante notar que a numeração dos nós em cada unidade funcional é peculiar a esta unidade. Todas as unidades funcionais têm o mesmo espaço de numeração de nós, isto é, de 1 a 2**NN.

Um outro aspecto a considerar é que as memórias CACHE não são associativas. Somente as memórias FILA DE ENTRADA são associativas, sendo que seu tamanho pode ser bem menor que as CACHE's. As operações associativas que ocorrem nestas memórias são as seguintes:

- Para os nós mapeados nesta unidade funcional, cuja posição no CACHE esteja vazia, procure algum pacote nas FILA DE ENTRADA's, começando pela cabeça da fila. Se algum pacote for encontrado, o mesmo deve ser transferido para o CACHE correspondente.

Já nos CACHE's, o trabalho é o seguinte:

- Um nó de duas entradas esta apto a executar se ambas posições nos CACHE's estiverem ocupadas com dados (pacotes). Um nó de uma entrada requer somente dados no CACHE da esquerda. O nó IT é diferente, mas igualmente trivial. Assim, seguindo um algorítmo qualquer que dê a todos os nós uma igual chance, deve ser escolhido um nó para ser executado, entre aqueles que estão aptos.

A alocação de nós nas unidades funcionais deve levar em conta se o nó em questão representa uma operação do grafo que será executada muitas vezes ou não. Assim, aqueles nós que representam as operações mais internas de construções iterativas, devem ser alocados a unidades funcionais que não sejam muito ocupadas com o trabalho de muitos nós, garantindo um pronto atendimento à execução.

Para facilitar o entendimento do mapeamento de nós na unidade funcional, segue-se um exemplo onde um trecho de um grafo, conforme a figura 16 A é mapeado em algumas unidades funcionais de uma máquina (figura 16 B).

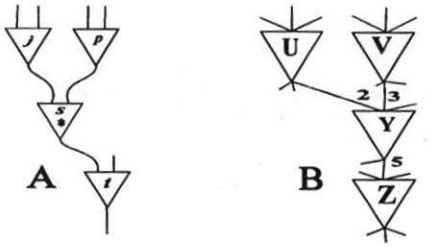

Figura 16 - Trecho de um Grafo e Mapeamento Correspondente

O nó *j* do grafo é mapeado na unidade **U** com o número **r**, *p* é mapeado em **V** com número **d**, *s* é mapeado em **Y** com número **q** e *t* é mapeado em **Z** com número **m**. Inicialmente, a unidade **Y** esta totalmente desocupada, com ambas filas e caches vazios. Na memória de programa da unidade **Y**, a palavra **q** (só mostrando os itens relevantes) será:

- TIPO DO NÓ = multiplicação

Descendente 1 (único):

- NÚMERO DA PORTA DE SAÍDA DO DMUX = 5
- NÚMERO DO NÓ NA UNIDADE DESCENDENTE = **m**
- ENTRADA = esquerda
- ANTI-FICHAS RECEBIDAS DO NÓ DESCENDENTE = 0
- RECEBEU UMA FICHA PARE = não

Entrada esquerda:

- NÚMERO DA PORTA DE ENTRADA DO MUX = 2
- NÚMERO DO NÓ NA UNIDADE ASCENDENTE = **r**
- MANDOU UMA FICHA PARE = não

Entrada direita:

- NÚMERO DA PORTA DE ENTRADA DO MUX = 3
- NÚMERO DO NÓ NA UNIDADE ASCENDENTE = **d**
- MANDOU UMA FICHA PARE = não

Com o decorrer da computação, no grafo, os nós *j* e *p* são disparados, e como resultado enviam dados para o nó *s* que deverá multiplicar os mesmos e enviar o resultado para seus descendentes. Na unidade **Y** (na máquina), a primeira ação que ocorre é o recebimento dos seguintes pacotes pelo MUX:

TIPO	NÚMERO DO NÓ NO DESTINO	ENTRADA	FICHA	DADO
3	q	esquerda	verde	2
3	q	direita	verde	486

A seguir, os referidos pacotes são enviados para os CACHES, já que as posições **q** de ambos estão vazias. Em sequência, a ALU perceberá que existe trabalho para fazer, pois as posições **q** dos CACHE's esquerdo e direito têm fichas e a memória de programa indica na posição **q** uma operação de multiplicação. A ALU retirará os dados dos CACHE's, realizará a operação de multiplicação, levando em conta o tipo de dado, e enviará o resultado (pacote) para a unidade **Z** pela porta de saída 5, para o nó **m**, com o seguinte pacote:

TIPO	NÚMERO DO NÓ NO DESTINO	ENTRADA	FICHA	DADO
3	m	esquerda	verde	972

Concomitantemente, as FILA DE ENTRADA's (esquerda e direita) serão pesquizadas associativamente a procura de algum novo pacote para o nó **q** que, se existir, será carregado no CACHE correspondente, na posição **q**.

5. Conclusões

O fato do modelo ser estático tem a desvantagem de não permitir reativação recursiva, que é, em essência, um processo serial, que não é exatamente o que se procura privilegiar aqui. A rigor, recursividade e paralelismo são inimigos naturais. Além disso, paralelismo também não convive bem com: a) subrotinas e código reentrante (da organização von Neumann), porque estas características têm um sentido de "afunilamento", isto é, processos independentes têm que compartilhar o mesmo recurso (o código), e b) ponteiros, porque estes transformam o espaço de endereçamento, quase todo, em uma única grande variável indexada, implicando na necessidade de sequenciamento de grande parte dos acessos à memória para garantia do determinismo. Em compensação, o modelo tem a vantagem de requerer, na implementação física, muito pouco trabalho de alocação de recursos em tempo de execução (fora a própria carga do programa).

O paralelismo discutido aqui e que é passível de ser explorado na arquitetura, cobre um largo espectro, isto é, vai desde o paralelismo existente nas operações aritméticas simples, em expressões aritméticas, até o paralelismo existente entre rotinas. Grande parte deste paralelismo brota naturalmente. Uma outra parte pode ser alcançada com pouco esforço de reestruturação dos grafos, aproveitando propriedades como associatividade e comutatividade das operações aritméticas simples. Finalmente, uma outra parte requer um esforço extra que é aquele relacionado com o desenrolamento de construções iterativas com dependências entre iterações. É neste tipo de paralelismo que o modelo permite grandes ganhos. Via de regra, em processos iterativos onde o resultado final é um somatório ou produtório, isto é, operações que têm a propriedade associativa, é possível reduzir o tempo de execução de N para log(N+1). Por exemplo, o fatorial de 8, demanda aproximadamente 7 unidades de tempo de multiplicação, se realizado iterativamente. No modelo, com um desenrolamento de 7 iterações, esse tempo seria reduzido para 3 unidades de tempo de multiplicação. Mantendo o desenrolamento de 7 iterações, o cálculo do fatorial de 16, que normalmente tomaria um tempo de 15 unidades de multiplicação, no modelo consumirá 6 unidades somente (2 iterações do grafo desenrolado).

Um outro aspecto interessante da organização proposta é que ela é completamente escalonavel, isto é, a máquina pode crescer a partir de uma única unidade funcional. Em adição, as "receitas" descritas para exploração de paralelismo (desenrolamento de construções iterativas e posterior reestruturação do grafo) são perfeitamente sistematizaveis, isto é, podem ser incorporadas em um compilador de uma linguagem bem estruturada qualquer, que é o meio julgado apropriado para programação da máquina.

Finalmente, é importante ressaltar que processamento paralelo é uma questão de: a) disponibilidade de unidades funcionais que possam trabalhar concorrentemente, assincronamente e autonomamente; e b) caminhos livres para que os dados possam rapidamente passar de onde são produzidos para onde são consumidos. Nesta proposta procurou-se atender estes requisitos.

Referências

[1] J. B. Dennis, "First Version of a Data Flow Procedure Language", Proc. Programming Symp., G. Goos and J. Hartmaris; Ed. Paris, France, 1974.

[2] J. R. Gurd, C. Kirkham, and I. Watson, "The Manchester Prototype Dataflow Computer", Communications of the ACM, vol. 28, n. 1, pp 34-52, Jan/85.
[3] Arvind and R.S.Nikhil, "Executing a Program on the MIT Tagged - Token Dataflow Architecture," IEEE Trans. on Computers, vol. 39, n. 3, pp 300-318, Mar/90.
[4] S. Sakai, Y. Yamaguchi, K.Hiraki, and T.Yuba."An Architecture of a Dataflow Single Chip Processor", Proc.16. Ann. Int. Symp. Computer Architecture, Jerusalem, Israel, pp. 46-53, May/89.
[5]R. S. Nikhil, G. M. Papadopoulos, and Arvind, "*T: A Multithreadded Massively Parallel Architecture", Proc. Ann. Intl. Symp. on Computer Architecture, pp 156-167, 1992.
[6] G. M. Papadopoulos, K. R. Traub, "Multithreading: A Revisionist View of Dataflow Architectures", Proc. Ann. Intl. Symp. on Computer Architecture, pp 342-351, 1991.
[7] C. R. Sonnenburg, "A Configurable Parallel Computing System", PhD thesis, University of Michigan 1974.
[8] W. Blume, R. Eigenmann, J. Hoeflinger, D. Padua, P. Petersen, L. Rauchwerger and Peng Tu, "Automatic Detection of Parallelism", IEEE Parallel & Distributed Technology, pp 37-47, Outono 1994.